NCC2-387

A General Utility for Plotting Functions on the Color Sun Workstation

Robert L. Brown

October, 1986

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 86.25

NASA Cooperative Agreement Number NCC 2-387

RIACS

Research Institute for Advanced Computer Science

A General Utility for Plotting Functions on the Color Sun Workstation

Robert L. Brown

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 86.25
October, 1986

ABSTRACT

Functionview is a general utility for displaying two dimensional line graphs on Sun workstations. Using this utility, one may write very simple programs that are concerned only with the computation at hand and may disregard any aspects concerning the generation of the graphical image. The program accepts as input a specification file, which defines a control panel into which the user can enter program parameters, a set on initial values for these parameters, the name of a supplied-user program to run, and the specification for the input to that program. This document contains all the information necessary for a Sun user to create programs and specification files easily, and to produce interesting and useful plots in very short order.

Work reported herein was supported in part by NASA Cooperative Agreement number NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

A General Utility for Plotting Functions
on the
Color Sun Workstation

Robert L. Brown

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 86.25
October, 1986

1. Introduction

The SUN Microsystems color workstation is a moderately powerful computer with the capability to perform moderately complex computations at reasonable speed, generate graphical representations of data moderately easily, and allow the creation of sophisticated user interfaces. Its shortcoming is that programs that generate graphical output must be, for the most part, hand crafted for each application. SUN provides no general tool that allows a user with no knowledge of graphical programming to create even simple data graphs, especially using the color capabilities of the model 160C. Because each application must be embedded in a program with a complicated section of code for handling the user interface and graphical output, the workstation may remain under utilized despite its capabilities. For example, in a simple application to display the motion of a swinging two-dimensional pendulum, the percentage of code that actually simulated the pendulum was 12%, whereas the code to handle the interactive front-end and the graphical backend was 88%.

The program described herein embodies that 88% in a single fixed program. The user is chartered only with writing a program to compute the desired function. The output of that program is simply pairs of numbers. The user also supplies a simple visual specification of a control panel that, when the utility is run, allows the user to modify the inputs to his simple computational program arbitrarily.

This utility is different from traditional graphics packages in that is provides a sophisticated interface to a computational program, and allows the user to specify the format of the input to that program. Instead of just using static data files as its input (which it is capable of doing), it invokes a user process, or program, provides it input from the user-specified control panel, and then collects and displays the results, or output, of that

process in a graphics window. On color workstations, colormap animation is provided to enhance the rendering of the plot.

This program is the first in a family of such programs. Subsequent programs add structure to the output of the computational programs, and hence add capabilites. We consider the general technique of coupling configurable user interface control panels and graphical backends with computational programs developed by users with no special graphics knowledge a sound one and plan to investigate its extensibility.

Figure 1 (at the end of the paper) show a sample run of *functionview*.

2. Usage

2.1. Setting Up

Functionview is the name of this utility. Once invoked, it creates a window on the workstation screen with four parts: a static control panel, a user-defined control panel, a messages window, and a plotting window. It has two inputs: a specification file and a computational program, both provided by the user. When it starts up, it reads the specification file and, based on its contents, builds the user-defined control panel. The user can then, using the control panels, set the input values for his program, and cause his program to be invoked. At this point, the utility collects the results and displays them graphically in a window, allowing for optional dynamic rescaling of the display based on the input data.

Thus the basic usage scenario is as follows: first, write a program, in C or Fortran, for example, and by doing so, define which variables in that program should be user definable inputs. Code the program so that it reads those input values from standard input using *scanf*, as follows:

```
scanf("%f", &myvariable);
```

or

```
scanf("%s", myvariable);
```

where *myvariable* is the variable to be read in. Since all numeric data values handled by *functionview* are inherently floating point, the **%f** format is always appropriate. However, variables that are integer in nature may be read using a **%d** format. This computation program should be coded so that its output is pairs of numbers, separated by white space (blanks, tabs, or newlines). Typically, the pairs of numbers will be generated one pair per line, with a blank between the numbers within a pair. Next, compile and link this program. Next, write the specification file, as described below. Third, run *functionview* with the name of the specification file as the first argument. All that is left now is to modify the input parameters, displayed on the control panel, and run the computational program by clicking on the screen **Go** button (using the leftmost mouse button).

The first input to *functionview* is the specification file which is divided into four parts. It is a simple text file, and the separators between the parts are lines containing nothing but two percentage signs, as with *yacc*. The four parts are, in order, a user control panel template, a set of initial values, a program name, and a program input specification.

user control panel template

%%
initial values for control panel variables
%%
computational program name
%%
computational program input specification

The following paragraphs describe each of the four sections in turn.

The first section of the specification file contains a template for the user control panel, in which *functionview* allows the user to modify the input values to his computational program. This section is a combination of literal text fields and variable fields. Variable fields are denoted by a string of alphanumeric characters preceeded by a dollar sign. All other fields are considered literal. Here is a sample control panel template:

Points: *$maxpoints* **Delta T(s):** *$deltat*
Gravity(m/s2): *$gravity* **Pendulum length(m):** *$length*
Initial Position X(m): *$initx* **Y(m):** *$inity*
Initial Velocity X(m/s): *$initvx* **Y(m/s):** *$initvy*

In this example, the italicized fields are the variables and the boldfaced fields are literals. *Functionview* copies the literal fields to the user control panel window unmodified, and replaces the variable fields with the values given in the initialization section. There is no hard limit on the number of literal and text fields that may appear in a control panel template, though using more than nine lines will cause the plotting window in the display to be clipped off at the bottom of the screen. The resulting control panel will look just like the template, except that variable names will be replaced by values. Variables in *functionview* may be either numerics or strings, as determined by their initial values described in the next paragraph.

The second section of the specification file contains initial values for the variable given in the control panel template section. These are of the form of assignment statements, where the left side is the name of a variable previously used in the control panel, and the right side is a constant. The only data types supported are strings and floating point numbers, though values without decimal points will always be displayed as if they were integers. If the righthand side of the assignment statement is a quoted string, then the variable on the lefthand side is forever tagged as being a string variable. The value stored internally for a string variable and the value displayed on the control panel does not have the enclosing quotation marks. Embedded quotation marks in the initialization section must be preceeded by a backslash. The backslash is not needed before embedded quotation marks on the control panel. Here is a sample initialization section:

```
deltat=0.1
maxpoints=2000
gravity=9.8
length=1.0
initx=0.5
inity=0.5
initvx=0.7
initvy=0.0
```

A sample showing a string variable is as follows:

```
file="/etc/hosts"
bins=32
```

Any uninitialized variables are tagged as being numeric and are left blank in the user control panel.

The third section of the specification file contains the name of a program to run when the user of *functionview* clicks on the **Go** button. This program, also provided by the user, is typically a compiled C or Fortran program, but may just as well be a UNIX utility such as *bc*.

The fourth section is formatted much like the first section. It contains a template for the input to the program named in the previous section. Variable fields in this section, that is, those beginning with dollar signs, are replaced by the current values of the variables in the user control panel. The values of string variables are copied in without enclosing quotation marks.

2.2. Operation

Once the specification file and computational program have been written, *functionview* can be invoked from a shell, giving the name of the specification file as the first and only argument.

functionview *filename*

Functionview will start up, create window on the screen comprising a static control panel (having the **Go** button and other items), a one-line messages window, the user control panel, and a plotting window.

2.2.1. Static Control Panel

The static control panel has several buttons, check boxes, and text fields. There is also a scrollbar and less frequently used items can only be accessed by scrolling the panel. The individual buttons can be activated by pointing at them with the mouse and clicking the leftmost button. While *functionview* is performing the requested operation, the button will appear grey meaning no other operation may be initiated. The buttons and their meaning are

Go When clicked, causes the user's computation program to be invoked and the input to it created by taking the values from the user control panel, formatting them as shown in the fourth section of the specification file. Once the computational program starts producing output, *functionview* starts reading it, a pair of numbers at a time, interpreting them as X (horizontal axis, positive left) and Y (vertical axis, positive up) values, and connecting successive pairs with lines.

Redraw

When clicked, causes *functionview* to redraw all the points (saved from the last time the **Go** button was clicked). This is useful if the plot limits (explained below) or color map information (also explained below) is changed.

Rescale

When clicked, causes *functionview* to compute the X and Y minimum and maximum values in the data and use them as the limits on the plot window. If the "Square" box is checked (explained below), the difference between the maximum

and minimum values for X and Y is constrained to be the same, resulting in an aspect ratio of one.

Quit

When clicked, *functionview* exits.

Print

When clicked, *Functionview* saves the computed X and Y values in a file named DATA in the local directory. The first line of this file is such that if the file is invoked as a shell script and the output is piped to a UNIX plot(3) filter, a hardcopy of the data will be produced.

Read

When clicked, *functionview* reads the file whose name follows the label "Initial values from file:" on the same line. Hence, the user must supply a file name before clicking this button.

The third row of the static control panel contains "toggles", which may be either on or off. Those in the *on* state are indicated by a checkmark in the small box next to their label. The toggles are

Auto Rescale

If set, causes *functionview* to rescale the plotted data automatically every so often if it has read and saved data points that are outside the current plot minimum and maximum.

Square

If set, the plot area will always be square, covering as much distance on the X axis as the Y.

Cycle

On a color workstation, data are drawn in rainbow colors (31 different colors). These colors are stored in a structure within the workstation called a *colormap*. *Functionview* plots each line in a particular color taken from the colormap, making a full pass through the map for each so-many data points, depending on a parameter described below. If the *cycle* toggle is set, *functionview* will rotate the colors through the colormap in a circular fashion, giving a simple sense of movement.

Reverse

If set, causes the color map animation to rotate in the opposite direction.

The fourth line of the control panel has the field **Points per color cycle**. The value stated on that line is the number of points *functionview* will plot for each full cycle through the colormap.

The last two normal lines of the static control panel state the current X and Y minimum and maximum in the plot area. The user may manually change these (then click on *Redraw*) or they can be computed by the rescale function.

In the hidden part of the control panel, accessible only by using the scroll bar, is a slider that can be used to set the speed of the color map rotation.

2.2.2. Messages Panel

The second panel on the window is a one line message area, where *functionview* displays status and error messages. The messages are meant to be self-explanatory, but

occasionally a cryptic one appears. A description of diagnostic messages is given below.

2.2.3. User Control Panel

As previously described, the user control panel is formed from the first section of the specification file. The user may manually change entries in this panel by pointing to the number to change, clicking the leftmost mouse button, and then using the standard character and line erase characters to edit the field.

2.3. Diagnostic Messages

Diagnostic messages that may appear in the messages window are as follows:

Terminated by non-numeric input: %s

This is caused by some input received from the user's program that cannot be parsed as a number. A typical instance is if an overflow occurs in the user's program and it tries to print the result. In that case, the output appears as the string "NaN".

Empty file name, cannot read.

This occurs when the user clicks the **Read** button but no file name has been entered by the input file prompt.

filename: *system error message*

This occurs when *functionview* tries to access a file and then receives an error return from the operating system. The name of the offending file and the text of the system error is shown.

3. Examples

3.1. Swinging Pendulum

The following example is one that simulates the motion of a two-dimensional pendulum. Figure 1 at the end of this paper show the results seen on the screen after running this example. Figure 2 shows the output generated by the **Print** button. First, the specification file is given, then the skeleton of the C program to perform the computation is given.

```
Points: $maxpoints                Delta T(s): $deltat
Gravity(m/s2): $gravity           Pendulum length(m): $length
Initial Position X(m): $initx     Y(m): $inity
Initial Velocity X(m/s): $initvx  Y(m/s): $initvy
%%
deltat=0.1
maxpoints=2000
gravity=9.8
length=1.0
initx=0.5
inity=0.5
initvx=0.7
initvy=0.0
%%
```

```
pend
%%
$deltat
$maxpoints
$gravity
$length
$initx
$inity
$initvx
$initvy
```

The third section contains the name of the compiled program that performs the simulation of the pendulum. In this example, this program expects eight numeric inputs, given in the order listed in the fourth section. The skeleton of the computational program, *pend.c*, is shown here. The fact that the names of the input variables in the program and the corresponding variables in the specification files are nearly identical is purely coincidental; there is no association between the names in the two files.

```
#include <stdio.h>
main()
{
        int i;
        int maxpoints;
        float delta_t, gravity, length, mass;
        float initx, inity, initvx, initvy;
        float x, y;

        scanf("%f",&delta_t);
        scanf("%d",&maxpoints);
        scanf("%f",&gravity);
        scanf("%f",&length);
        scanf("%f",&initx);
        scanf("%f",&inity);
        scanf("%f",&initvx);
        scanf("%f",&initvy);

        for( i = 0 ; i < maxpoints ; i++ ) {

/* COMPUTE NEW POSITIONS x & y,then print */

                printf("%g %g\n", x, y);

        }
}
```

3.2. Using bc

This example shows how the UNIX utility *bc* can be used with *functionview*. This example specification file generates and plots a simple sine curve. Notice that the *bc*

program is in the specification file itself, rather than being read in from another source. Because *bc* is so slow, this program takes a couple of minutes to run.

```
Number of points: $points
Number of cycles: $cycles
%%
points=100
cycles=2
%%
bc -l
%%
for(i=0 ; i<$points ; i++ ) {
   r=i/$points*3.141592654*2*$cycles
   r
   s(r)
}
```

3.3. Inline Shell Scripts

Arbitrary shell scripts may be encoded in *functionview* specification files. To do so, the program section of the file should contain simply **sh** and the input section should contain the body of the shell script. Some care must be taken to assure that any variables used in the shell script do not conflict with the names of the variables used in the control panel. *Functionview* will copy through unchanged any strings beginning with a dollar sign that are not variables defined in the control panel.

This example plots a histogram of file sizes. It reads the directory named in the control panel and, using *awk*, builds a histogram based on the size of the files. The output of a sample run is shown in Figure 3.

```
HISTOGRAM OF FILE SIZES
Directory: $dir
Histogram Bin Size: $binsize
Maximum File Size: $max
%%
dir="."
binsize=64
max=4096
%%
sh
%%
ls -l $dir |
awk 'BEGIN  {maxbin=0}
    $4<=$max {bin = int($4/$binsize)
              hist[bin]++
              if(bin>maxbin)
                    maxbin = bin
             }
  END        {for(i=0 ; i<=maxbin ; i++) {
                    hist[i] += 0;
```

```
        print i,hist[i]
        print i+1,hist[i]
}
print maxbin+1,0
print int($max/$binsize)+1,0
}'
```

4. Conclusions

We have shown that a simple plotting utility program can be written in a way that an application programmer may make use of the powerful, control panel-oriented user interface programming style of the Sun workstation without being concerned with the details of screen management necessary for such programs. The approach was to embody all those details into a single spearate program which then invokes the user's program and collects and displays its output. Future workk will involve extending this paradigm to more sophisticated graphics applications.

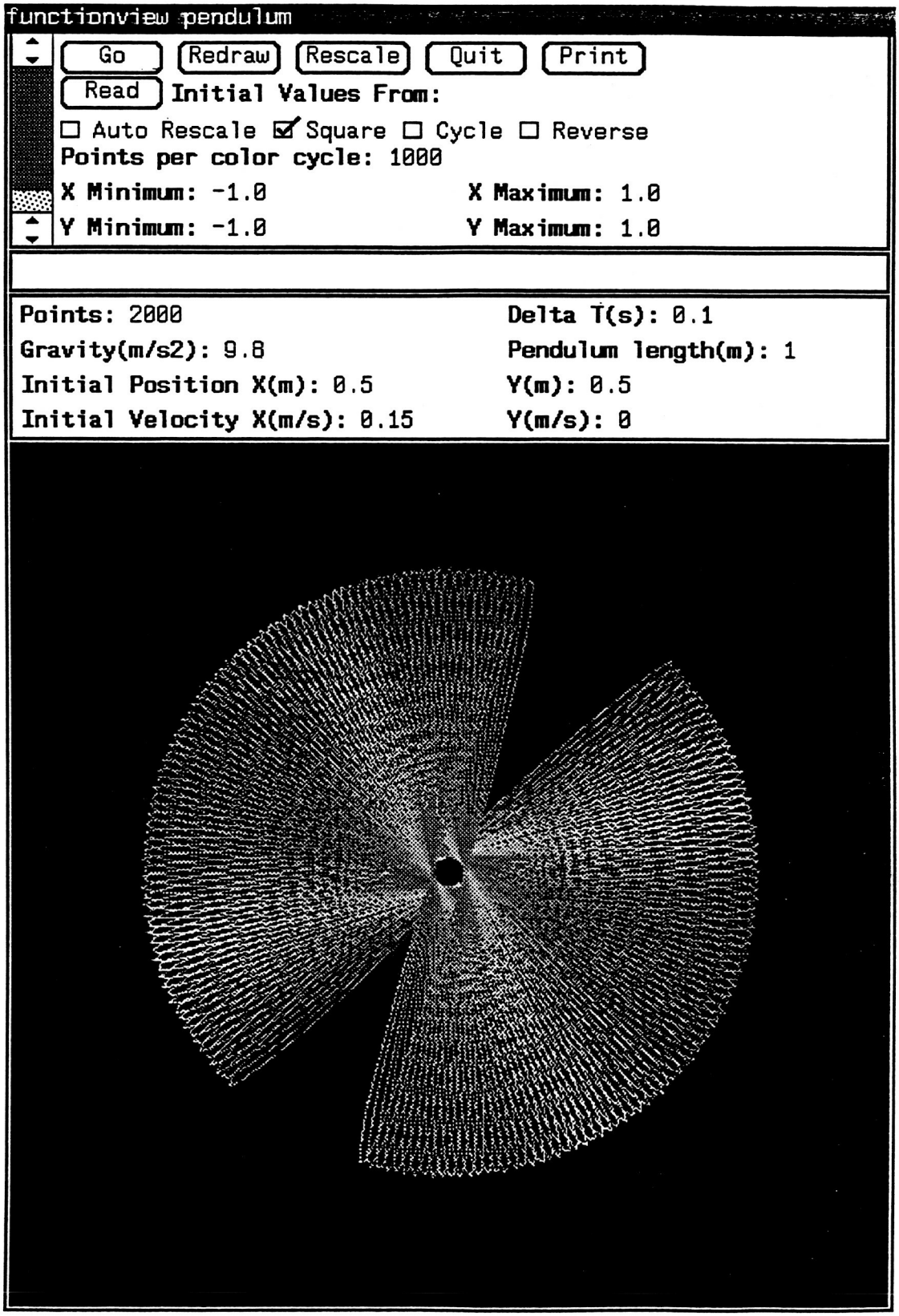

Figure 1. Sample functionview display.

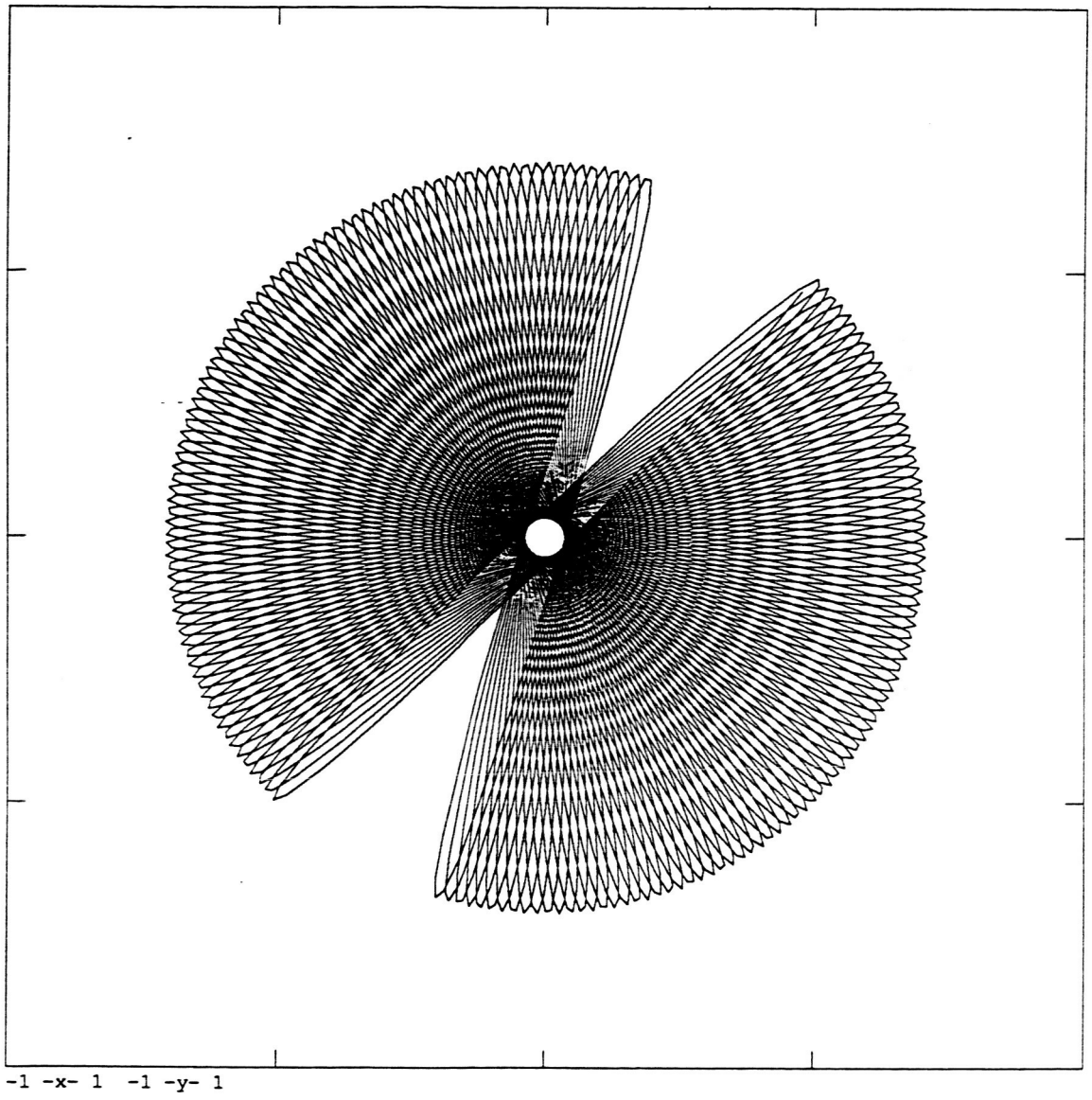

Figure 2. Results generated by the **Print** button.

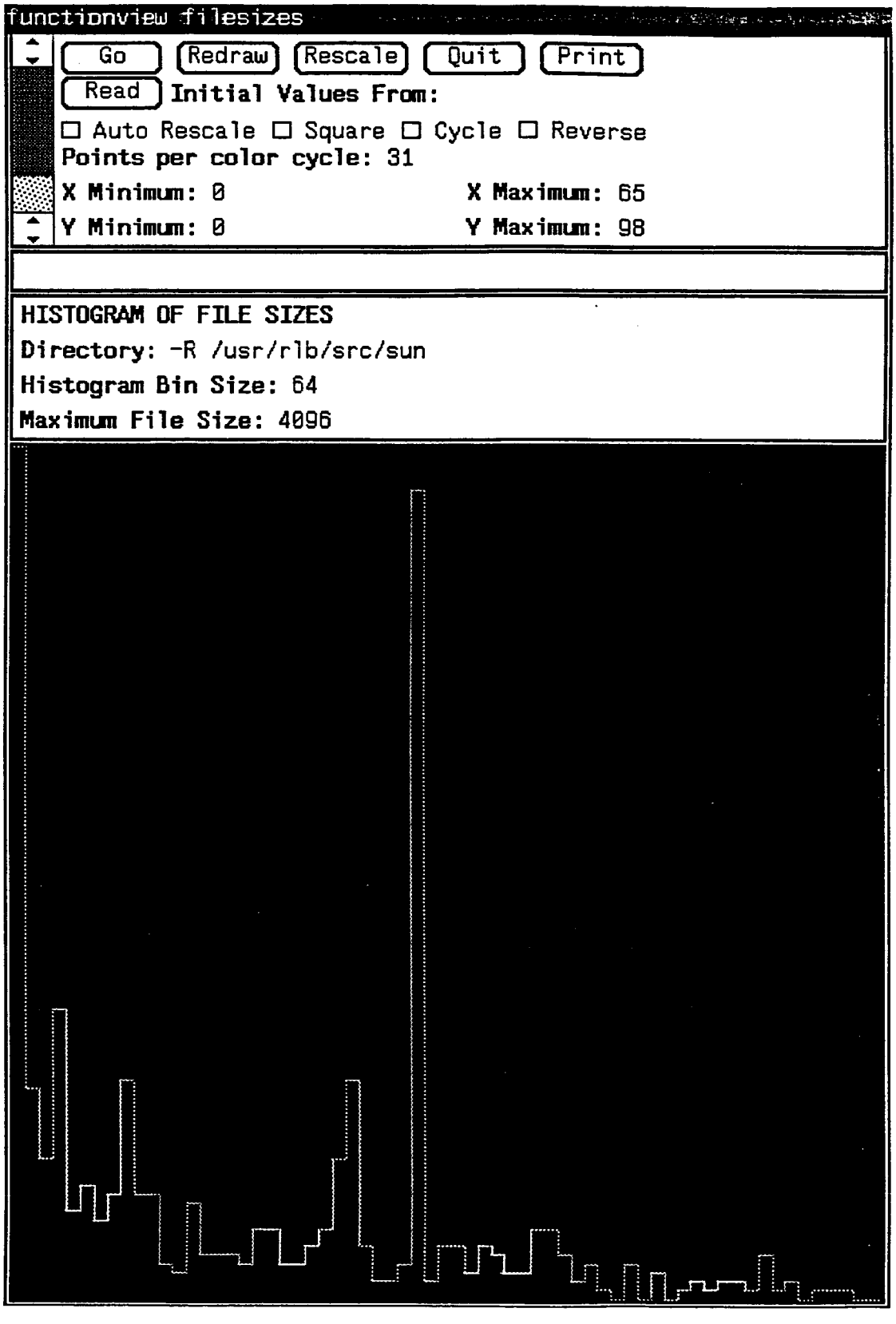

Figure 3. Output of Shell Script Sample.